EP 33239 (3)

F2

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT).

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
13 June 2002 (13.06.2002)

PCT

(10) International Publication Number
WO 02/46909 A1

(51) International Patent Classification[7]: G06F 7/02

(21) International Application Number: PCT/US01/47932

(22) International Filing Date: 7 December 2001 (07.12.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/254,277 7 December 2000 (07.12.2000) US

(71) Applicant *(for all designated States except US)*: **WEBPUTTY** [US/US]; 2 West Santa Clara Street, 2nd Floor, San Jose, CA 95113 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **HERBERT, Charles, St. John, III** [US/US]; 54 Citation Drive, Los Altos, CA 94024 (US). **SHULMAN, Semyon** [US/US]; 26 Cadiz Circle, Redwood City, CA 94065 (US).

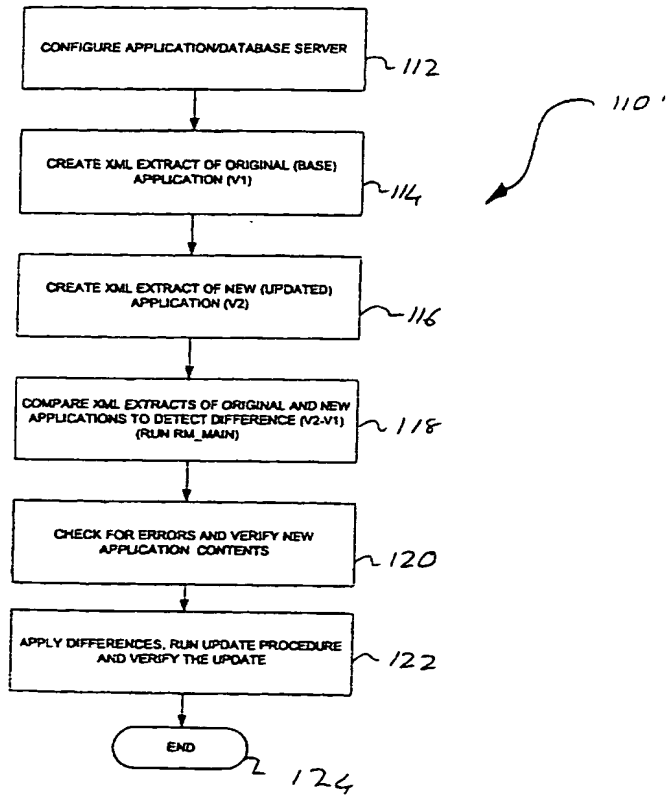
(74) Agents: **MALLIE, Michael, J.** et al.; Blakely, Sokoloff, Taylor & Zafman LLP, 7th floor, 12400 Wilshire Boulevard, Los Angeles, CA 90025 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM),

[Continued on next page]

(54) Title: AUTOMATICALLY DEPLOY AND UPGRADE AN APPLICATION BASED ON MARKUP LANGUAGE APPLICATION DEFINITION



(57) Abstract: A method to deploy software application includes creating a structured document defining the software application (114). The structured document is automatically is utilized to deploy the software application (122). The structured document may comprise a markup language document, such as a XML document (114, 116). The structured document may describe multiple tiers of the software application (e.g., a presentation tier, a logic tier and a data tier) and may defined any one or more of data, logic and interfaces of the software application. In one exemplary embodiment, the creating of the structured document includes extracting the structured document from first metadata describing the software application (114).

WO 02/46909 A1

European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report*

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

A METHOD AND SYSTEM AUTOMATICALLY TO DEPLOY AND UPGRADE AN APPLICATION BASED ON AN APPLICATION DEFINITION

FIELD OF THE INVENTION

The present invention relates generally to the field of software development and, more specifically, to a method and system automatically to deploy and/or upgrade case of application utilizing an application definition.

BACKGROUND OF THE INVENTION

In traditional environments, a software application may exist as a collection of many pieces and components (e.g., compiled code and data) stored on a multiplicity of service in a deployed environment. An Information Technology (IT) manager overseeing such a deployed environment is required to track the version and location of each component of such a software application, including the server on which each component is installed. In order to move, copy or upgrade the software application, the IT manager tested at the required to halted each server on which a component is to stored, copy each and every component of the software application, and then re-deploy such components on a target server.

SUMMARY OF THE INVENTION

In one embodiment, the present invention provides a method to deploy a software application. The method includes creating a structured document defining the software application. The structured document automatically is utilized to deploy the software application. The structured document may comprises a markup language document, such as a XML document for example

The structured document may describe multiple tiers of the software application (e.g., a presentation tier, a logic tier and a data tier) and may define any one or more of data, logic and interfaces of the software application.

In one exemplary embodiment, the creating of the structured document includes extracting the structured document from first metadata describing the software application.

The automatic deployment of the software application may include automatically

upgrading the software application, the method including performing a comparison between the software application and an upgrade software application, generating a differences structured document describing differences between the software application and the upgrade software application, and modifying the software application into at least partial conformance with the upgrade software application utilizing the differences structured document.

The automatic upgrading of the software application may optionally be transactionally performed in order to facilitate the upgrade without halting execution of the software application.

The comparison between the software application and the upgrade software application, in one exemplary embodiment, includes performing a comparison between the structured document defining the software application and a further structured document defining the upgrade software application.

The comparison between the software application and the upgrade software application may optionally include performing a comparison between objects of the software application and the upgrade software application, and generating the differences structured document based on the comparison.

The modification of the software application may include modifying first metadata describing the software application utilizing in the differences structured document.

The modification of the software application may, for example, include addition, subtraction or changing of objects comprising the software application.

The differences structured document may be a differences XML document.

A respective differences structured document may also be generated for each object of the group of objects for which the comparison detected a difference.

Other features of the present invention will be apparent from the accompanying drawings and from the detailed description that follows.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is illustrated by way of example and not limitation in the figures of the accompanying drawings, in which like references indicate similar elements and in which:

Figure 1 is a block diagram illustrating a system, according to an exemplary embodiment of the invention, to deploy (e.g., an initial or upgrade deployment), a software application.

Figure 2 is a block diagram illustrating various exemplary functions that may be included within the release management system of the flexible server according to an exemplary embodiment of the present invention.

Figure 3 is a process flow chart illustrating a method, according to an exemplary embodiment of the present invention, of building a target software application based, for example, on a source software application.

Figure 4 is a flow chart illustrating a method , according to an exemplary embodiment of the present invention, of building a database portion of the interoperability and persistence tier 24 of a flexible server 12.

Figure 5 is a process flow illustrating further details regarding the method illustrated in Figure 4.

Figure 6 is a process flow illustrating a method, according to an exemplary embodiment of the present invention, of performing an upgrade operation to original software application so as to conform the original software application to an upgrade software application.

Figure 7 is a flow chart illustrating a method , according to an exemplary embodiment of the present invention, of performing an update (or upgrade) process to create a database update package for delivery to an existing flexible server installation.

Figure 8 is a flow chart illustrating a method, according to an exemplary embodiment of the present invention, whereby a compare operation, may be implemented.

Figure 9 is a block diagram providing an alternative representation of the exemplary update process described with reference to Figure 7 and illustrates various

exemplary data structures and processes that may be involved in one embodiment of an application update process.

Figure 10 shows a diagrammatic representation of machine in the exemplary form of a computer system within which a set of instructions, for causing the machine to perform any one of the methodologies or operations discussed above, may be executed.

DETAILED DESCRIPTION

A method and system automatically to deploy a software application based on an application definition are described. In the following description, for purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be evident, however, to one skilled in the art that the present invention may be practiced without these specific details.

Figure 1 is a block diagram illustrating a system 10, according to an exemplary embodiment of the invention, to deploy (e.g., an initial or upgrade deployment), a software application. The system 10 includes a flexible server 12 that, in the exemplary embodiment, runs on top of, and augments, a platform stack (e.g., the Microsoft .NET platform). The flexible server 12 is shown to run on top of respective web, application, and database servers 14, 16, and 18, which in turn sit on top of an operating system 20.

The flexible server 12 is an n-tier server that enables developers to create and deliver flexible business systems, and also allows users to effect changes and updates in the rules and processes in a near real-time. As a result, a system manager can define the condition and user access rules to insure security, and allow only authorized users to access business and application logic. Utilizing the flexible server 12, a user is able to build and modify applications by describing databases, attaching business rules, and logic to data elements, and designing interfaces (e.g. access screens for service integration, and interoperability interfaces) for data elements.

The flexible server 12 facilitates the development and delivery of software applications by conceptually separating the business logic of a software application from the source code. As illustrates in **Figure 1**, the business logic is defined and stored as an application definition 22, which may also be referred to as metadata. The application definition 22 is a programming language-neutral format that stores the definition of a software application (e.g. logic, data models, and rules) in a database. The application

definition 22, in one embodiment, defines all aspects of a software application supported by the flexible server 12, including properties, behaviors, logic, relationships, architecture, and presentation of the software application. The flexible server 12 statically and dynamically instantiates, or implements, a software application based on such definitions. If changes to a software application are required, a user with appropriate permission can reflect such changes in the application definition 22, and the flexible server 12 will dynamically implement the relevant changes.

In one embodiment, the flexible server 12 is logically separated into three tiers, namely an inoperability and persistence tier 24, an engine tier 26 and an interface tier 28. Each of the tiers of the flexible server 12 utilizes the application definition 22 for definition of function. In this way, the application definition 22 may conceptually be viewed as a "blue print" for a particular software application. As shown in **Figure 1**, the flexible server 12 may build and continuously update a structured document description, in the exemplary form eXtensible Markup Language (XML) documents 30 that describe the functionality and dependencies of a software application. Such XML documents may be persistently stored in a database, or dynamically cached in a memory.

The separation of the application definition from source code by the application definition 22 provides a number of benefits. First, by storing logic, data models, interfaces, etc. in the application definition 22, there is a single source of information regarding the characteristics of the object of a software application. This enables a change, or multiple changes, to be automatically propagated throughout a software application. Without a single application definition 22, a change may involve multiple time-consuming changes to each tier of a software application, any of which could introduce inconsistencies and errors. In short, the application definition 22 facilitates the implementation of inherited properties across multiple tiers of a software application.

Second, the application definition facilitates referential integrity throughout a software application because, in one embodiment, properties and logic defined in the application definition 22 automatically reference other items within the flexible server 12 upon which they depend. These references are enforced, thereby providing consistency. Third, the application definition 22 provides users with insights into items and dependencies on a particular rule, assuring that changes can be made with relative safety and ease. The application definition 22 provides an understanding of

relationships between objects of different tiers, which is traditionally challenging as such an understanding requires review of a large volume of source code.

Fourth, according to one embodiment of the present invention, a software application can be deployed (e.g., an initial deployment or and upgrade deployment) by exporting the latest version of the software application as a structured document (e.g., XML) representation. The flexible server 12, and more specifically a release management system 32 incorporated within the flexible server 12, performs a deferential between the exported representation of the latest version of the software application and a structured document representation of an existing version of the application. The release management system 32, as will be described in further detail below, then upgrades a deployed system (e.g., a flexible server 12 implementing a software application) in place. Similarly, a system can be rolled back to a previous version.

Further, the application definition 22 is independent of any specific platform, as the flexible server 12 runs on and interfaces with an underlying platform and takes care of platform and language-specific details, including modifications in platforms and languages.

Further, by defining a software application in the application definition 22, the flexible server 12 reduces the amount of code required to be written by an application developer.

Referring to the exemplary embodiment of the system 10 shown in **Figure 1**, the flexible server 12 is shown to be logically separated into the three tiers discussed above, with the functionality of each tier being defined by a corresponding application definition 22. While the flexible server 12, in one exemplary embodiment, runs on top of industry-standard technology, each tier corresponds with a standard enterprise-class server, all of which in turn function on top of an operating system 20.

Dealing now specifically with the tiers implemented within the flexible server 12, the interoperability and persistence tier 24 allows application to interoperate with external application, objects, web services, and databases, to maintain persistent state for a software application implemented by the flexible server 12. The interoperability and persistence tier 24, in one embodiment, makes use of facilities of a relational database, with native support for SQL Server, Oracle, and DB2. The flexible server 12 can automatically discover properties of external objects, data sources, and web services and, once discovered, the flexible server 12 can access and interface with these external sources to exchange data. The flexible server 12 also creates application database

objects based on definitions within the application definition 22. In addition to base tables, an application definition engine (not shown) generates procedures, views, constraints, and keys. These mediate interaction with base tables, and maintain integrity and consistency of a database.

Turning now to the engine tier 26, this tier 26 utilizes, in the exemplary embodiment, a distributed component architecture (MTS/COM+), and provides advanced transaction management.

The engine tier 26 is shown to include complex business objects 33 that combine data and behavior to deliver functionality of a particular software application. Generic business objects derive data from, for example, tables or objects in the interoperability and persistence tier 24, and implement CREATE, READ, UPDATE, DELETE (CRUD) behavior. The complex business objects 33 can also be specialized via a business logic engine 34.

The business logic engine 34 provides the mechanism to execute custom business logic. Business logic may be defined in terms of rules that are triggered based upon events, such as data insert, update, and delete. Rules may also take the form of custom methods implemented in complex business objects.

The engine tier 26 is also shown to include permissions 36 that in, one embodiment, implement a security model encompassing many aspects of application security. For example, authentication is based on a user name/password pair, which can be integrated into operating system authentication or make use of a native authentication mode. Access permissions are granted on a role basis, and provide multiple levels of granularity, from access to high-level constructs (e.g. menus), all the way down to field-level security. In addition to access control, the flexible server 12 provides built-in support for segmenting data by organization, so that information can be shared among organizations, or safeguarded within such organizations. Auditing facilities 38 are also available to track application activity.

The flexible server 12 can, in one embodiment, also dynamically serve application in different human languages to address the needs of multi-language applications. An exemplary manner in which this function may be provided is described in U.S. Patent No. 6,018,742.

As described above, the engine tier 26 also includes a release management system 32 that, in the exemplary embodiment, provides automated export, import, and upgrade of applications by, in one embodiment, packaging the application definition 22

into a structured document (e.g., XML) representation. The release management system 32, in one embodiment using the XML representations of two software application versions, provides the capability to detect the differences between the versions of the software application. In this way, a software application can be upgraded automatically and in place.

The engine tier 26 may also provide runtime management (not shown) that provides a mechanism to manage various aspects of the flexible server 12 at runtime. For example, the runtime management functions may include cache management, application management, and audit management.

Moving on now to the interface tier 28, a presentation interface 42 enables the flexible server 12, in one exemplary embodiment, to present a standard browser-based user interface. Web pages are automatically rendered by the presentation interface 42 appropriate for a requesting browser type, which may be dynamically determined.

The presentation interface 42 may include a library of user interface (UI) controls that provide behavior based on properties specified by the application definition 22 (e.g., default values, validations, permissions, labels, etc.). In addition to familiar controls such as text inputs, check boxes, selects, images, and links, the flexible server 12 has a set of higher-level UI abstractions (e.g., grids, menus, and filters). Beyond such UI building blocks, the flexible server 12 also facilitates the implementation of UI paradigms, such as query-update and master-detail.

The presentation interface 42 also includes a page builder function that allows pages to be generated dynamically from markup language (e.g., HTML) templates combined with controls enabled by the application definition 22. Multiple target browsers, technologies, and clients may be supported (e.g., Internet Explorer, PocketPC, Netscape, and HTML 32).

A template library within the presentation interface 42 contains markup language (e.g., HTML) templates for screen generated and updated based on an application definition 22. A HTML editor may be utilized to customize the design of an interface by modifying such templates. Customized style sheets, client scripts, and DHTML may also be included within the template library.

The interface tier 28 is also shown to include a XML exchange engine 44 that, in the exemplary embodiment, enables software applications implemented by the flexible server 12 to consume and provide complex data to and from outside partners in the XML format.

Web services 46 within the flexible server 12 allow the flexible server 12 to consume and publish web services. A reports function 48 within the interface tier 28 facilitates interfacing with commercial reporting tools, such as Microsoft Access and Crystal Report. Utilizing familiar report writers, application developers can accordingly deliver customized, professional reports within a software application supported by the flexible server 12.

As will be appreciated from the above, the flexible server 12 can be used to create complex new applications, or to replace, upgrade and enhance existing software applications. A software application supported by the flexible server 12 can be utilized to extend the functionality and capability of an existing application, essentially "enhancing-parallel" via data and function sharing.

Figure 2 is a block diagram illustrating various exemplary functions that may be included within the release management system 32 of the flexible server 12 according to an exemplary embodiment of the present invention. The release management system 32 is shown to include an application export function 50 that operates to build, and continuously update, a structured document representation, in the exemplary form of a XML extract, of the application definition 22 (or metadata) of a particular software application.

An application import function 52 operates to deploy a software application based on a structured document, in the exemplary form of a XML extract, defining (or describing) a software application.

An application compare function 54 operates to compare application definitions 22 (or metadata) describing or defining applications, and then to generate a differences structured document (e.g., a differences XML document) indicating the differences between the application definitions 22. In one embodiment, the comparison of the application definitions 22 may be performed by comparing XML extracts representing the respective application definitions 22.

An application upgrade function 56 operates, in a manner to be described in further detail below, to upgrade an application definition 22 (or metadata) based, for example, on differences detected between application definitions 22 of an existing version of a software application and an upgrade version of the software application. An application delete function 58 operates to delete an applications metadata, and actual data of the software application based on certain criteria.

Figure 3 is a process flow chart illustrating a method 60, according to an exemplary embodiment of the present invention, of building a target software application 62 based, for example, on a source software application 61. The method 60 may, for example, be performed to clone the source software application 61 as the target software application 62 executing on the same, or a different, platform as the source software application 61. It should be noted that the method 60 operates to build multiple tiers within the target software application 62.

As illustrated in **Figure 3**, the source software application 61, as implemented by the flexible server 12, may be viewed as conceptually comprising an application definition (or metadata) 22, and multiple tiers of source code 23 (e.g., the tiers 24, 26, and 28 described above in reference to **Figure 1**). The application export function 50 of the release management system 32 is shown to access the application definition 22 of the source software application 61, and generate a structured document 30 (e.g., a XML extract or export) that comprises a representation of the application definition 22. The generation of the structured document as a XML document, for example, is advantages in that a XML document is readily communicated between multiple objects, is platform independent, and may be universally interpreted and understood by applications that are able to decode the appropriate tags within the XML document.

The structured document 30 is then utilized by the application import function 52 of the release management system 32 to regenerate an application definition 22 for the target software application 62.

Figure 4 is a flow chart illustrating a method 70, according to an exemplary embodiment of the present invention, of building a database portion of the interoperability and persistence tier 24 of a flexible server 12. **Figure 5** is a process flow illustrating further details regarding the method 70.

At block 72, a new database is created by a user to hold a release. Specifically, a Microsoft Word SQL Enterprise Manager 82 may be utilized to create the new database.

At block 74 of the method 70, the user creates an ODBC data source for the new database on the same server as which the new database was created at block 72.

At block 76, the newly created database is populated with data and metadata as specified by an application definition 22 chosen to define the particular software application. In an exemplary embodiment, a stored procedure is run to perform the population of the database a block 74. The stored procedure assembles all products (and versions) that have been associated with a common customer name and release, for

example.

The execution of an exemplary stored procedure (build_main) is indicated at block 88 of **Figure 5**, which is illustrated as being performed as part of the building of a release database at block 90.

At block 78 of the method 70 illustrated in **Figure 4**, a simple test is performed of the new build database 86. The method then ends at block 80.

Further details regarding an exemplary manner in which the method 70 may be performed by a user are described in a U.S. Provisional Application 60/254,277 entitled "Release Management System and Method of Operating The Same", filed December 7, 2000, and incorporated herein by reference.

Figure 6 is a process flow illustrating a method 100, according to an exemplary embodiment of the present invention, of performing an upgrade operation to original software application 102 so as to conform the original software application 102 to an upgrade software application 104. As illustrated in **Figure 6**, each of the original and upgrade software applications 102 and 104 include a respective application definition 22 and source code 23. The application export function 50 of the release management system 32 is shown to access the application definition 22 of each of the applications 102 and 104 to generate an original structured document (e.g., a XML export or extract) based on the application definition 22 of the original software application 102 and an upgrade structured document (e.g., a XML export or extract) based on the application definition 22 of the upgrade software application 104. It will accordingly be appreciated that the original and upgrade structured documents are representations of at least a portion of the corresponding application definitions 22.

The application compare function 54 of the release management system 32 then performs a comparison (or differences) operation between the original structured document and the upgrade structured document to generate a differences structured document (e.g., a XML document) 106 that reflects differences between the original structured document and the upgrade structured document. The ability of the release management system 32 to perform the comparison operation between two structured documents in the exemplary form of XML documents is advantageous in that such a comparison can be performed in a computationally efficiently manner.

The application upgrade function 56 of the release management system 32 then accesses the differences structured document 106, and upgrades the application definition 22 of the original software application 102 based on the indicated differences

so as to conform the application definition 22 of the original software application 102 to the application definition 22 of the upgrade software application 104.

While the method 100 has been described with reference to performing an upgrade to the application definition 22 of a software application, it will readily be appreciated that the method 100 is also useful to upgrade the source code 23 of the original software application 102 into conformance with the source code 23 of the upgrade software application 104.

Figures 7-9 illustrate various aspects of an upgrade process to upgrade solely the database of an interoperability and persistence tier 24 of a flexible server 12, according to an exemplary embodiment of the present invention. Specifically, **Figure 7** is a flow chart illustrating a method 110, according to an exemplary embodiment of the present invention, of performing an update (or upgrade) process to create a database update package for delivery to an existing flexible server 12 installation. The method 110 illustrates in **Figure 7** is initiated by a user wishing to perform the update process.

At block 112, an application and/or database server is configured. Specifically, the user chooses a server at which to create the update package. Various support files, in order to support a flexible server 12, are then put in place on the selective server.

At block 114, a XML extract (or export) representing the original software application 102 is created. At block 116, a XML extract of a new (or updated) application is created. As described above, the operations performed at blocks 114 and 116 are performed by the application export function 32 of the 50 of the release management system 32.

At block 118, the application compare function 54 compares the XML extracts of the original and new software applications to detect differences. In an exemplary embodiment, a procedure entitled rm_main is run to detect these differences. Further details regarding the operations that may be performed by this exemplary procedure are described below with reference to **Figure 8**.

At block 120, a check for errors is performed. Specifically, two log files (e.g., rm_error.log and rm_status.log) are created to reflect any errors detected during the c ompare operation at block 118. At block 120, the user may then also verify the new database contents. Specifically, upon completion of the compare operation (e.g., by the rm_main procedure), a table directory in a development database will contain the following code:

1. update scripts (e.g., .sql files);

2. fmt files;

3. differences files, captured as structured document (e.g., XML files);

4. dtd files required explaining DTD files that include DTD files that explain and describe the XML files;

5. a modification file (e.g., rm_mods_t. dat) that contains a shortlist about all database objects and information about the additions and modifications that have detected by the compare process. Each entry in the modification file may be formatted <object name> , <type>, <action>, where the various actions may be indicated as being add, modify, special or none actions.

Examples of an exemplary modification file, as well as exemplary possible actions and exemplary object types, are described below:

WF_CHILD_BO_LINKS_V,VIEW,M

WF_PARENT_ BO_LINKS_V,VIEW,M

WF_RULE_CONDITION_BO_SUB_V,VIEW,M

WF_RULE_CONDITION_BO _V,VIEW,M

DEFINE_PROCEDURE,GLOBAL_SP,S

DENORMALIZE_SCREEN_ITEMS,

CREATE_PROC_WF,GLOBAL_SP,S

COLUMN_TYPES,MASTER_TABLE,N

COLUMN_TYPES_L,MASTER_TABLE,N

TABLES,MASTER_TABLE,N

TABLES_L,MASTER_TABLE,N

COLUMNS, MASTER_TABLE,N

COLUMNS_L,MASTER_TABLE,N

TABLE_COLUMNS,MASTER_TABLE,N

TABLE_COLUMNS_L,MASTER_TABLE,N

UNIQUE-KEY_COLUMNS,MASTER-TABLE,N

TUNING_KEY_COLUMNS,MASTER_TABLE,N

tables,DATAUS,M

columns,DATA,M

procedures,DATA,M

codes,DATUS,M

rm_deliverables, DATA,M

rm_deliverables,DATAUS,M

The possible actions are:

Action	Description
A	Add
M	Modify
N	None, information only
S	Special (information that is used to keep track of metadata dependencies that determine the order in which tables will be updated)

The different types are:

Type	Explanation
MAN	Manual stored procedure
VIEW	Manual view
MQ	Manual Query procedure
GLOBAL_SP	Manual stored procedures dependent on metadata tables
LOCAL_SP	Manual and application tables
MASTER_TABLE	A master table used by the system
TABLE	Physical table
DATA	Data
DATA[2-character language suffix]	Language-dependent data (for example, DATAUS)

A user may utilize the above described modification file to check on the update process, and confirm that various changes have been changes have been captured. For example, a newly added procedure would be indicated as follows in the modification file.

NAME	TYPE	ACTION
Procedure_Mine	MAN	A

A user can verify that language-dependent data has been modified:

NAME	TYPE	ACTION
Rm_deliverables	DATAUS	M

Returning to the flow chart illustrated in **Figure 7**, following the error check and verification operations at block 120, the application upgrade function 56 of the release management system 32 is invoked at block 122 to apply the differences reflected in the differences structured document (e.g., a collection of XML files) by running an update procedure that implements the application upgrade function 56, and by verifying the update. In one exemplary embodiment, an update procedure (e.g., rm_target_main) accomplishes the following:

1. Executes a pre-update script.

2. Recompiles all manual stored procedures and manual view.

3. Imports data differences for all master tables.

4. Updates the structure of all affected tables (modified or new).

5. Imports data differences for all master tables.

6. Creates affected foreign keys (apply changes and creates new keys).

7. Creates/recreates all generated stored procedures.

8. Executes the post_upg.sql script.

9. Recreates all generated stored procedures.

10. Reports all differences (updates performed and errors) to log files.

exec rm_target_main

<target_dsn>,<sa_login>,<sa_pwd>,<language>,<user_login>,<mode>

<target_dsn>	Data Source name for the target database being updated.
<sa_login>	SQL server admin login.
<sa_pwd>	SQL server admin password.
<language>	2-character language code from the Language table.
<user_login>	Login for the user who will first log in to validate the new release. User should have master role privileges.
<mode>	Procedure run mode: V: verbose – everything is printed to the screen. D: debug – expanded output to the screen. R: release – limited output to the screen.

Running the rm_target_main procedure creates two files in the DatabaseDev directory, rm_error.log and rm_status.log. these files will contain any errors incurred during execution.

The update performed at block 122 may be verified, for example, for developing and running test scenarios on key functionality components of the upgraded application.

The method 100 then ends at block 124.

Figure 8 is a flow chart illustrating a method 160, according to an exemplary embodiment of the present invention, whereby the compare operation, performed at block 118 of **Figure 7**, may be implemented. In one embodiment, the method 160 is executed by as an rm_main procedure that at least partially implements the application compare function 54 of the release management system 32.

The method 160 commences at block 162 with the identification of new or modified data objects and changes in tables. Each of the operations performed at blocks 162-170 of the method 160 are directed to locating and identifying modifications in a software application supported by the flexible server 12. In one embodiment, the operations performed at blocks 162-170 are performed by comparing an application definition 22 of an existing (or base) software application with the application definition 22 of an upgrade (or update) software application. In one embodiment, the respective application definitions 22 are reflected in XML exports (or extracts) 30. The find operations performed at blocks 162-170 furthermore each generate respective structured documents (e.g., XML files) that record the modifications to the respective data structures and objects. In one embodiment, these may conveniently be termed "differences structured documents".

At block 172, the compare procedure proceeds to make a number of changes following completion of the location and identification operations performed at blocks 162-170. At blocks 172-176, the compare procedure proceeds to make a number of changes to the existing (or base) database. Specifically, at block 172, all stored procedures are recompiled, at block 174 metadata tables are updated with modifications to match data in the old and new software applications 102 and 104, and at block 176, structural changes are made for metadata and seeded data tables are propagated from tables of the upgrade software application 104 to the tables of the original software application 102. The method 160 then ends at block 178.

A description of an exemplary compare procedure (e.g., rm_main) is provided immediately below:

exec rm_main

<old_server_name>,<old_db_name>,<old_user_group>,<target_dsn>,<sa_login>, <sa_pwd>,<login>,<language>,<mode>

<old_server_name>	Name of the database server with the base database.
<old_db_name>	Name of the base database.
<old_user_group>	User group on base database.
<target_dsn>	Data Source Name for the old database.
<sa_login>	SQL server admin login.
<sa_pwd>	SQL server admin password.
<login>	Login for the user who will first log in to validate the new release.
<language>	2 character language code from the Language table.
<mode>	Procedure run mode: V: verbose – everything is printed to the screen. D: debug – expanded output to the screen. R: release – limited output to the screen.

Figure 9 is a block diagram providing an alternative representation of the exemplary update process described above with reference to **Figure 7** and illustrates various exemplary data structures and processes that may be involved in one embodiment of an application update process.

It will be appreciated that updating (or upgrading) the database of an original software application is somewhat more complex than an original deployment because the contents of the existing database should be retained.

For this reason, **Figure 9** illustrates, in one embodiment, SQL Enterprise Manager 132 generating a copy of a prior release (e.g., the original software application 102) 136.

A build procedure (e.g., a build_main procedure 138) is shown in **Figure 9** to be utilized to build a current release 142 (e.g., an upgrade software application 104) of a base software application (e.g., the original software application 102). The development

database 144 is shown to provide input to the build process at block 140.

Figure 9 also illustrates that the prior release 136 and the current release 142 provide input to a differences detection process 146 (e.g., performed by the rm_main 148) that operates to produce structured documents (e.g., XML documents 154) that reflect data differences. The XML files 154 may reflect differences in both so-called "master" tables that constitute metadata (or an application definition) and differences in "seeded" data tables that contain actual data. In one embodiment, a XML document may be generated for each so-called "master" table and for each "seeded" data table. The differences detection process 146 is also shown to output a file 156 listing all objects changed (e.g., rm_mods_t.dat). The file 156 describes each object type, as well as the modification action for the each relevant object.

The file 156, the XML files 154, and manually created SQL scripts 152 (which operate to alter the scheme, if required, of any "master" tables) are then packaged.

The actual installation, for example, may involve the following steps, some of which require user input.

1. Copy all components to the server.

2. Create the temporary table listing object changes from file 156 (e.g., rm_mods_t.dat).

3. Create or replace all manual stored procedures and views based on differences list (e.g., rm_mods_t).

4. If any "master" tables have changed, upgrade them via the manual scripts prepared.

5. If any "master" tables have data differences, import the differences from their XML files.

6. If any "seeded" table's structure changed, upgraded the table to restructure it and regenerate stored procedures and views.

7. For any "seeded" tables with data differences, import data from the table's

XML file.

8. Create stored procedures as appropriate.

9. Delete stored procedures, views and files that are no longer needed.

Further details regarding exemplary embodiments of the various functions that are incorporated within the release management system 32 are provided below. It will again be appreciated that the present invention is not limited to add these exemplary implementation details.

METADATA HIERARCHY TABLE

In exemplary embodiments described below, Metadata Control Codes (MCCs) are utilized to implement these exemplary embodiments. In these embodiments, the metadata that constitutes an application definition 22 may include a table METADATA_HIERARCHY_W to define the relationship amongst Metadata Control Codes. A Metadata Control Code, in one embodiment, is associated with each into within many tables of the metadata that constitutes an application definition 22 for the purpose of classifying the data within the redundant entry. Multiple Metadata Control Codes may be selected for inclusion within a particular software application. The relationship between Metadata Control Codes may be parent-child relationship, where the child Metadata Control Code is defined to be enterable for the instance of an object that is a child to a Parent Object with the Parent Metadata Control Code.

APPLICATION EXPORT FUNCTION (5 0)

PURPOSE:

Implement Export Application (e.g., based on Metadata Control Code).

SCOPE:

In this exemplary embodiment, on Metadata Control Code, export seeded data to XML.

ASSUMPTIONS:

- If data table is seeded, it should have an MCC column.

- MCC from table_columns table for MCC column should match the MCC being exported.

HARD CODED:

- Insertion of TABLES, PROCEDURES into rm_mods_t.

ITEMS INVOLVED:

Name	Type	Action
Export_XML_W	Stored Procedure	Add
Create_PROC_XML_EXPORT	Stored Procedure	Existing
RM_CREATE_XML_INI_W	Stored Procedure	Add

IMPLEMENTATION DETAILS:

1. EXPORT_XML
    a. Create directory
    b. Copy .fmt files from tables directory
    a. Remove existing *.xml and *.dtd files from @mcc_tables_directory
    b. Remove existing *.xml and *.dtd files from @tables_directory
    c. Remove existing *.dat files
    d. Create temporary table rm_mods_t
    e. Insert into rm_mods_t MCC record
    f. Insert into rm_mods_t MASTER_TABLE records
    g. If MCC is dependent upon other MCC, insert "P" record into rm_mods_t
    h. Create export_xml_lang cursor with all implemented languages
    i. Create export_xml_tables cursor with all metadata and seeded tables having a metadata_control_code column and LOOP
        i. If exists data in current table with current metadata_control_code then insert into rm_mods_t (there is data to be exported to XML)
        ii. If exists current table in rm_mods_t
            1. If current table = 'TABLES'
                a. insert into rm_mods_t all new tables
            2. If current table = 'PROCEDURES'
                a. insert into rm_mods_t all new views and manual procedures

b. copy views and manual procedures to mcc_tables_directory

3. Create _xx procedure using CREATE_PROC_XML_EXPORT
   a. Denormalize table columns
   b. Create temp table _t
   c. Create temp table #TT
   d. Use temp table #TT to create _xx procedure
   e. Drop _xx procedure
   f. Create _xx procedure using EXEC_TT
4. Run xx procedure without language to create .xml and .dtd in tables_directory
   a. Declare a variable for each column
   b. Delete temp table _t
   c. Build xml file in _t table
   d. Use tag _TABLE
   e. Select all from tables_s into cursor tables_tc – use ' ' for language dependent fields
   f. Convert all columns to xml except language dependent fields
   g. WRITE_STRING_FILE to write .xml to file only if data exists
   h. Delete temp table _t
   i. Build dtd file in _t table
   j. WRITE_STRING_FILE to write .xml to file only if data exists
5. Insert into rm_mods_t for each language
6. Run xx procedure for each implemented language to create .xml and .dtd in tables_directory
   a. Declare a variable for each column
   b. Delete temp table _t
   c. Build xml file in _t table
   d. Use tag _LANGUAGE

e. Select all from tables_s into cursor tables_tc with matching language id – use language dependent fields

f. Convert table_code, language dependent fields, last_update_datetime, metadata_control_code to xml

g. WRITE_STRING_FILE to write .xml to file only if data exists

h. Delete temp table _t

i. Build dtd file in _t table

j. WRITE_STRING_FILE to write .xml to file only if data exists

7. Move .xml and .dtd from tables directory to mcc_tables_directory
8. Create .INI configuration file
9. Write rm_mods_t table to file

APPLICATION COMPARE FUNCTION (54)

PURPOSE:

Detect and report the differences between two sets of XML files describing seeded data for the two versions of the WP App.

ASSUMPTIONS:

- The ini file is the driver for the whole operation of running the diff. It should be error free for the XML diff to work correctly.
- Process is to be run on DB server containing latest version of specified software application identified by unique Metadata Control Code value (up to 10 chars)
- DB App XML extract had run and two sets of XML extract files (or exports) are available. One set represents the earlier (e.g., original) version and second (e.g., upgrade) set represents the latest version that we will need to upgrade to. Each set contain XML extracts of all seeded data associated with the App MCC value. Each

seeded table will be represented by *table_code*.xml, *table_code*.dtd files. If there is a language dependent data, one *table_code_language_code*.xml file for each language will be present as well.

- Special handling will be required for extracts across different application versions. .

SCOPE:

- All information is to be written to the TABLES_DIRECTORY\MCCValue\Diff??

- Report all activities to be performed to the log rm_*MCCvalue_RevTag*_diff_status..log and errors to rm_*MCCvalue_RevTag*_diff_err..log.
  - Log Format: [Section xxxxxx] Specify Action being taken: Time
    - For EXAMPLE:
    - [Section 1000.000] XML Diff started at: Feb 22 2001 1:29PM
    - [Section 2000.000] [Time: Jul 10 2001 03:07PM] Finding data for deletes
    - [Section 3000.000] [Time: Jul 10 2001 03:07PM] Comparing s.xml
    - [Section 4000.000] [Time: Jul 10 2001 03:07PM] Comparing si.xml
    - ...
    - [Section 9000.000] XML Diff finished at: Feb 22 2001 1:45PM
  - RM_MAIN Example
    - [Section 51000.000] rm_main started at: Feb 22 2001 1:29PM
    - [Section 51000.000] old server name hpbrio45 specified
    - [Section 51000.000] old db name Upgrade_Old specified
    - [Section 51000.100] Executing rm_base
    - ...
    - [Section 51000.000] rm_main completed at: Feb 22 2001 1:29PM

- Summary of the compare will need to be written to rm_mods_t.dat and rm_deletes_t.dat files.

Rm_mods_t.dat file is a comma-delimited file with the following 3 data elements:

- DB object name (up to 128 chars)
- DB object type (up to 10 chars)
- DB object action (1 char)

Rm_deletes_t.dat is a comma-delimited file with the following 4 data elements:

- Dep_seq (int)
- Table_code (up to 128 chars)
- Field1 (up to 255 chars)
- Field2 (up to 255 chars)

- Remove all existing files from the diff directory, if any.

RULES FOR XML DIFFERENCE

- Report the MCC value of the something application to rm_mods_t.dat file in the following manner – **MCCValue, MCC,M** (e.g. WPINV,MCC,M). Report in rm_mods_t.dat MCCValues that WP App MCC is dependent on based on the data in the new metadata_hierarchy_w XML extract: **PARENT_APPLICATION_ID_W_APPLICATION_MCC_TAG_W*value*, MCC, P** (PARENT_APPLICATION_ID_W_APPLICATION_MCC_TAG_W is available in the extract.

  There should be at least one parent named **PRODUCT** for each application. Also, if there is more than one parent, then no parent value should be empty (fatal error).

- Report master tables to rm_mods_t in the following manner:
  - COLUMN_TYPES,MASTER_TABLE,N
  - COLUMN_TYPES_L,MASTER_TABLE,N
  - TABLES,MASTER_TABLE,N
  - TABLES_L,MASTER_TABLE,N
  - COLUMNS,MASTER_TABLE,N

- COLUMNS_L,MASTER_TABLE,N
- TABLE_COLUMNS,MASTER_TABLE,N
- TABLE_COLUMNS_L,MASTER_TABLE,N
- UNIQUE_KEY_COLUMNS,MASTER_TABLE,N
- TUNING_KEY_COLUMNS,MASTER_TABLE,N

Scenarios for the existence of Tables.xml in the two directories:

S. No.	Does File exist in Old Directory	Does File exist in New Directory	Is there any entry for the File in the ini file	Action
1.	No	Yes	Yes	Table_code from each row in new file will need to be reported to rm_mods_t.dat with name = TABLE_CODE value, type = 'TABLE', and action = 'A'.
2.	Yes	Yes	Yes	Do all the actions specified in the next section below.
3.	Yes	No	No	Do nothing
4.	Yes	Yes	No	Do nothing.

- Obtain quantitive tables information by comparing new and old tables.xml (if available):

  - Any old table_code that do not exist any longer in the new will need to be reported to rm_mods_t.dat with name = TABLE_CODE value, type = 'TABLE', and action = 'D'.
  - Any new table_code that do not exist in the old will need to be reported to rm_mods_t.dat with name = TABLE_CODE value, type = 'TABLE', and action = 'A'.

- Any table_code that exists in old and new will need to be reported to rm_mods_t.dat with name = TABLE_CODE value, type = 'TABLE', and action = 'M'

Scenarios for the existence of Procedures.xml in the two directories:

S. No.	Does File exist in Old Directory	Does File exist in New Directory	Is there any entry for the File in the ini file	Action
1.	No	Yes	Yes	Procedure_code for each row in a new file with procedure_type of 'MAN' or 'VIEW' will need to be reported to rm_mods_t.dat with name = PROCEDURE_CODE value, type = PROCEDURE_TYPE, and action = 'A'. All procedure_codes of a procedure_type 'MAN' will need to be reported to rm_mods_t.dat with name = PROCEDURE_CODE value, type = 'LOCAL_SP', and action = 'S'
2.	Yes	Yes	Yes	Do all the actions specified in the next section below.
3.	Yes	No	No	Do nothing
4.	Yes	Yes	No	Do nothing.

- Obtain quantitive manual procedures and views information by comparing new and old procedures.xml (if available):

- Any old procedure_codes with procedure_type of 'MAN' or 'VIEW' that do not exist any longer in the new will need to be reported to rm_mods_t.dat with name = PROCEDURE_CODE value, type = PROCEDURE_TYPE, and action = 'D'.
- Any new procedure_codes with procedure_type of 'MAN' or 'VIEW' that do not exist in the old will need to be reported to rm_mods_t.dat with name = PROCEDURE_CODE value, type = PROCEDURE_TYPE, and action = 'A'.
- Any procedure_codes with procedure_type of 'MAN' or 'VIEW' that exist in old and new will need to be reported to rm_mods_t.dat with name = PROCEDURE_CODE value, type = PROCEDURE_TYPE, and action = 'M'
- All procedure_codes in the new of a procedure_type 'MAN' will need to be reported to rm_mods_t.dat with name = PROCEDURE_CODE value, type = 'LOCAL_SP', and action = 'S'

- Detect deleted information on the following XML files:

- si.xml, s.xml, ui_properties_w, ui_classes_w, ui_themes_w, menu_items_xml (in this order)

- dependency_sequency can be incremented
- If there is a record in OLD but not in NEW, then insert a line in rm_deletes_t (<dep_seq>,<table_code>,<field1>,<field2>) *field2 is optional dependending on table_code.
- If deletes found while comparing si.xml, insert a line in rm_deletes_t.dat
  - For example → **1,si,screen_code,field_identifier**
- If deletes found while comparing s.xml, insert
  - Example → **2, s, screen_code,** (notice the comma at the end...it represents field2 is NULL)
- If deletes found while comparing ui_properties_w.xml, insert
  - Example → **3, ui_properties_w, ui_property_code,**
- If deletes found while comparing ui_classes_w.xml, insert
  - Example → **4, ui_classes_w, ui_class_code_w,**
- If deletes found while comparing ui_themes_w.xml, insert
  - Example → **4, ui_themes_w, ui_theme_code_w,**

- If deletes found while comparing menu_items.xml, insert
  - Example → **4, menu_items, menu_code,**

- Compare all XML file sets to detect the actual data differences:

- **Scenario Discussion (examples):**

S. No.	Scenario	Action
1.	If a particular table is specified as a language dependent table in the ini file and there doesn't exist a physical file corresponding to each language (specified in the ini file). (e.g., lets say that the ini file says that the languages are **US, ES, DE** and for a table **Codes** the new directory just contains Codes.xml and Codes_US.xml).	Write a warning in the Error Log saying that the particular language dependent file doesn't exist. e.g., as per our example write that Codes_ES.xml and Codes_DE.xml don't exist.
2.	If a particular table is specified in the ini file but it doesn't exist physically in the directory.	Write a warning in the Error Log saying that the particular xml file doesn't exist. Continue processing. Reason: A table could be seeded but there might not be any data in that table yet.. So, the appropriate .xml does not have to exist..

- If the base table doesn't exist, then the language dependent tables also won't exist. For example, if there is no Codes.xml, then there won't be any Codes_US.xml or Codes_DE.xml etc. (fatal error)

- If there is a certain seeded table files set in the old and it is not on new, no information need to be written to rm_mods_t.dat

- If there is a certain seeded table files set in the new and it is not on old, copy the entire new set to the diff directory and report the following information to the rm_mods_t.dat file:
  TABLE_CODEvalue,DATA,A
  TABLE_CODEvalue,DATAUS,A
  TABLE_CODEvalue,DATAGE,A
- If there is a certain seeded table files set in the new and in the old and the dtd structure has changed, copy the entire new set to the diff directory and report the following information to the rm_mods_t.dat file:
  TABLE_CODEvalue,DATA,M
  TABLE_CODEvalue,DATAUS,M
  TABLE_CODEvalue,DATAGE,M
- If there is a certain seeded table files set in the new and in the old and the dtd structure has not changed, compare the data between new and old and write rows in new that do not exist in old or rows in new when row exists in new and old and data for this row was changed. Last_Update_Datetime can be excluded from compare. Create the appropriate files in the diff directory (table_code.dtd, table_code.xml, table_code_US.xml, table_code_GE.xml). If differences detected only for the base tables files, language data (table_code_us.xml, table_code_ge.xml) do not need to be written and visa-versa. The following information needs to be reported to the rm_mods_t.dat file:
  TABLE_CODEvalue,DATA,M (changes are detected in the base file)
  TABLE_CODEvalue,DATAUS,M (changes are detected in the us file)
  TABLE_CODEvalue,DATAGE,M (changes are detected in the ge file)

ERROR HANDLING (FATAL CASES):

In all the following exemplary error conditions, further processing may be stopped.

S. No.	Condition	Handling
1.	The directory for the ErrorLog file doesn't exist.	Error
2.	The directory for the StatusLog file doesn't exist.	Error

3.	The ErrorLog file doesn't exist.	Error
4.	The StatusLog file doesn't exist.	Error
5.	The directory specified as having the old files doesn't exist.	Error
6.	The directory specified as having the new files doesn't exist.	Error
7.	The ini file doesn't exist at the location specified.	Error
8.	The rm_mods_t.dat file doesn't exist in the old directory	Error
9.	The rm_mods_t.dat file doesn't exist in the new directory	Error
10.	Can't load the xml file	Error
11.	Instantiation of DOM failed	Error
12.	Error while writing to a file.	Error
13.	There is no parents app for this app or at least one parent app is not PRODUCT	Error
14.	Base table doesn't exist and language dependent table does exist. Check for both the new and the old directories.	Error

In all the following warning conditions a warning message is written in the Error Log and further processing goes on:

S.No.	Condition	Handling
1.	If a particular table is specified as a language dependent table in the ini file and there doesn't exist a physical file corresponding to each language (specified in the ini file). Check for both the new and the old directories.	Warning
2.	If a particular table is specified in the ini file but it doesn't exist physically in the directory. Check for both the new and the old directories.	Warning.

APPLICATION COMPARE FUNCTION (E.G. , XMLDIFF) (54)

This section specifies the code flow for a differences component of the compare function 54, according to exemplary embodiment of the present invention, conveniently labeled the utility XMLDiff .

The exemplary differences component starts with the _tmain() function which does the basic checks like whether the number of arguments is proper or not, all the desired directories and files are existing or not.. It then deletes all the files existing in the diff directory except the Status and Error logs.

Validation (Initial)

Using the ini file get the names of the different languages, the differences component gets the names of the special tables (for future delete operations), get the unique keys for the tables and get the language dependencies for the tables. It creates a seeded tables vector that will specify that whether the given table is language dependent or not.

Write standard lines in Rm_mods_t.dat

the differences component to then writes some hard-coded lines in the rm_mods_t.dat and then starts the actual processing:

Checks on Metadata_hierarchy_w table

The differences component calls ExtractSpecifiedColumnValueForEachRow() which checks the value of the element <PARENT_APPLICATION_ID_W_APPLICATION_MCC_TAG_W> for all ROWS in the file *Metadata_hierarchy_w.xml* existing in the New directory and then writes the value along with the strings MCC,P(e.g., if the value is PRODUCT it writes, PRODUCT, MCC,P) in the rm_mods_t.dat file.

The differences component to then starts a for loop according to the seeded tables vector created earlier:

Processing: In a loop:

- Check whether it is a special table and if it's so then set the deleted flag.

- Check whether its ***Tables.xml*** and if its so then call ProcessQuantitativeTablesData(). The function ProcessQuantitativeTablesData() checks for the existence of the new and old Tables.xml.

  If the old Tables.xml doesn't exist, then it copies the new Tables.xml in the diff directory and writes all the Table Codes of the new file with the action as 'A' in the rm_mods_t.dat file.

  Else it calls CompareTablesXMLRowbyRow(). CompareTablesXMLRowbyRow() compares the two files by internally calling LoadValMaps() and CompareValMaps().It then writes any discrepancies in the two files to rm_mods_t.dat according to the Tables rules specified earlier.

- Check whether its ***Procedures.xml*** and if its so then call ProcessQuantitativeProceduresData(). The function ProcessQuantitativeProceduresData() checks for the existence of new and old Procedures.xml and then it calls ProcedureDifference() specifying to it whether the old Procedures.xml exists or not.

  If the old file doesn't exist, then ProcedureDifference() copies the new Procedures.xml to the diff directory and writes all the Procedure Codes of the new file having the Type as "MAN" or "VIEW" with the action as "A" in the rm_mods_t.dat file.

  Else ProcedureDifference() compares the two files by internally calling LoadValMaps2() and then doing the line-by-line comparison. It then writes any discrepancies in the two files to rm_mods_t.dat according to the Procedures rules specified earlier.

- Then it calls ProcessOneTable() for all the files. One of the parameter of ProcessOneTable() is the deleted flag that we set earlier for the special tables. This function also checks for the existence of new and old files.

If the old file doesn't exist it just copies the new xml and dtd files into the diff directory and writes a line specifying the action as "A" in the rm_mods_t.dat file.

Else it calls RowByRowDifference(). RowByRowDifference() calls LoadValMaps2() and then it compares the two files and writes the discrepancies into the rm_mods_t.dat according to the rules . It checks that whether there are any rows added/deleted/modified. Also it creates new xml and dtd files as per general rules specified earlier.

For special tables (having the deleted flag set), we detect the deleted information and report it into the rm_deletes_t.dat file as per specified format.

- Again depending on the language dependency of the file, it calls ProcessOneTable() for each language.

TESTING:

The various Test Scenarios and their results are put in a separate excel document **XMLDIFF_TEST.xls** which is in the same folder as the current document.

APPLICATION DELETE FUNCTION (58)

PURPOSE:

Implement Delete function based, for example, on Metadata Control Code

SCOPE:

Based on a single Metadata Control Code, dynamically delete all related database objects.

- For each data record, dynamically detect and delete all dependent records from any table
- Dynamically drop tables, procedures, views that is associated with given MCC
- Do not delete child records that are outside of given MCC

- Detect and track each referential integrity failure
- Report all committed or failed transactions to a text file for user reference

REQUIREMENTS / ASSUMPTIONS:

- If a table has a column named "ID", check for dependent data else delete from table.
- If a table does not have an ID column, the data will just be deleted with an assumption that they are no dependent data.
- If a table does have an ID column, then it checks for dependent data.
- Process may take several minutes to complete, depending on number of records and complexity
- Complexity: The bottom most child shouldn't be more than 12 levels deep from the parent

HARD CODED:

- "TABLES" (table)
- "PROCEDURES" (table)
- "TABLE_COLUMNS" (table) – uses 'table_columns' to delete wuz_base
- "WUZ_BASE" - Deletes wuz_base table manually for performance reasons.

ITEMS INVOLVED:

Name	Type	Action
RM_Remove_App_By_MCC	Stored Procedure	Add
RM_Remove_Parent_Child	Stored Procedure	Add

NOTE:

- If procedure is deleting a row while one or more of its child tables do not have a Metadata Control Code column, the child (dependent) data related to the parent row from child (dependent) tables will still be deleted.

IMPLEMENTATION DETAILS:

1. RM_Remove_App_By_MCC procedure finds all tables that have a column "METADATA_CONTROL_CODE"

2. If MCC is PRODUCT or CUSTOM, the procedure will generate an error and exits
3. Create temp table #tables_tmp (name, type)
4. Declare Cursor and Select all tables that contain Metadata Control Code column
    a. If current table is 'TABLES', drop all dependent objects and finally drop tables that is associated with the given MCC (uses temp table '#tables_tmp' to drop)
    b. If current table is 'PROCEDURES', if exists drop each procedure
    c. Delete from wuz_base and wuz_transactions manually that references each dropping procedure
    d. If current table is 'TABLE_COLUMNS', delete from wuz_base manually that references table_column_id
    e. Execute rm_remove_parent_child_w for each record that needs deletion
    f. RM_Remove_Parent_Child_w
        i. Declare Cursor and select all dependent tables
        ii. For each record, recursively execute RM_Remove_Parent_child_w until there are no dependent tables/data.
        iii. Once there are no more dependent records, delete the parent record
    g. Close Cursor

Write to log files if there were no errors.

Figure 10 shows a diagrammatic representation of machine in the exemplary form of a computer system 200 within which a set of instructions, for causing the machine to perform any one of the methodologies discussed above, may be executed. In alternative embodiments, the machine may comprise a network router, a network switch, a network bridge, Personal Digital Assistant (PDA), a cellular telephone, a web appliance or any machine capable of executing a sequence of instructions that specify actions to be taken by that machine.

The computer system 200 includes a processor 202, a main memory 204 and a static memory 206, which communicate with each other via a bus 208. The computer system 200 may further include a video display unit 210 (e.g., a liquid crystal display

(LCD) or a cathode ray tube (CRT)). The computer system 200 also includes an alphanumeric input device 212 (e.g., a keyboard), a cursor control device 214 (e.g., a mouse), a disk drive unit 216, a signal generation device 218 (e.g., a speaker) and a network interface device 220.

The disk drive unit 216 includes a machine-readable medium 222 on which is stored a set of instructions (i.e., software) 224 embodying any one, or all, of the methodologies or functions described herein. The software 224 is also shown to reside, completely or at least partially, within the main memory 204 and/or within the processor 202. The software 224 may further be transmitted or received via the network interface device 220. For the purposes of this specification, the term "machine-readable medium" shall be taken to include any medium that is capable of storing, encoding or carrying a sequence of instructions for execution by the machine and that cause the machine to perform any one of the methodologies of the present invention. The term "machine-readable medium" shall accordingly be taken to included, but not be limited to, solid-state memories, optical and magnetic disks, and carrier wave signals.

Thus, a method and system automatically to deploy a software application based on an application definition have been described. Although the present invention has been described with reference to specific exemplary embodiments, it will be evident that various modifications and changes may be made to these embodiments without departing from the broader spirit and scope of the invention. Accordingly, the specification and drawings are to be regarded in an illustrative rather than a restrictive sense.

CLAIMS

What is claimed is:

1. A method to deploy a software application, the method including:

creating a structured document defining the software application; and

utilizing the structured document automatically to deploy the software application.

2. The method of claim 1 wherein the structured document comprises a markup language document.

3. The method of claim 2 wherein the markup language document comprises a XML document.

4. The method of claim 1 wherein the structured document describes multiple tiers of the software application.

5. The method of claim 4 wherein the multiple tiers of the software application included a presentation tier, a logic tier and a data tier.

6. The method of claim 1 wherein the structured document defines any one or more of data, logic and interfaces of the software application.

7. The method of claim 1 wherein the creating of the structured document includes extracting the structured document from first metadata describing the software application.

8. The method of claim 1 wherein the automatic deployment of the software application comprises automatically upgrading the software application, the method including performing a comparison between the software application and an upgrade software application, generating a differences structured document describing differences between the software application and the upgrade

software application, and modifying the software application into at least partial conformance with the upgrade software application utilizing the differences structured document.

9. The method of claim 8 wherein the automatic upgrading of the software application is transactionally performed in order to facilitate the upgrade without halting execution of the software application.

10. The method of claim 8 wherein the comparison between the software application and the upgrade software application includes performing a comparison between the structured document defining the software application and a further structured document defining the upgrade software application.

11. The method of claim 8 wherein the comparison between the software application and the upgrade software application includes performing a comparison between objects of the software application and the upgrade software application, and generating the differences structured document based on the comparison.

12. The method of claim 8 wherein the modification of the software application includes modifying first metadata describing the software application utilizing in the differences structured document.

13. The method of claim 8 wherein the modification of the software application includes addition, subtraction or changing of objects comprising the software application.

14. The method of claim 8 wherein the differences structured document comprises a differences XML document.

15. The method of claim 8 including generating a respective differences structured document for each object of the group of objects for which the comparison detected a difference.

16. A system to deploy a software application, the system including:

an export function to create a structured document defining the software application; and

a deployment function to utilize the structured document automatically to deploy the software application.

17. The system of claim 16 wherein the structured document comprises a markup language document.

18. The system of claim 17 wherein the markup language document comprises a XML document.

19. The system of claim 16 wherein the structured document describes multiple tiers of the software application.

20. The system of claim 19 wherein the multiple tiers of the software application included a presentation tier, a logic tier and a data tier.

21. The system of claim 16 wherein the structured document defines any one or more of data, logic and interfaces of the software application.

22. The system of claim 16 wherein the export function extracts the structured document from first metadata describing the software application.

23. The system of claim 16 wherein the deployment function includes an upgrade function that is automatically to upgrade the software application, and wherein the upgrade function is to perform a comparison between the software application and an upgrade software application, to generate a differences structured document describing differences between the software application and the upgrade software application, and to modify the software application into at least partial conformance with the upgrade software application utilizing the differences structured document.

24. The system of claim 23 wherein the upgrade function is to transactionally upgrade software application in order to facilitate the upgrade without halting execution of the software application.

25. The system of claim 23 wherein the upgrade function is to perform a comparison between the structured document defining the software application and a further structured document defining the upgrade software application.

26. The system of claim 23 wherein the upgrade function is to perform a comparison between objects of the software application and the upgrade software application, and generating the differences structured document based on the comparison.

27. The system of claim 23 wherein the upgrade function is to modify first metadata describing the software application utilizing in the differences structured document.

28. The system of claim 23 wherein the rate function is to automatically a cup addition, subtraction or changing of objects comprising the software application.

29. The system of claim 23 wherein the differences structured document comprises a differences XML document.

30. The system of claim 23 wherein the upgrade function is to generate a respective differences structured document for each object of the group of objects for which the upgrade function detected a difference.

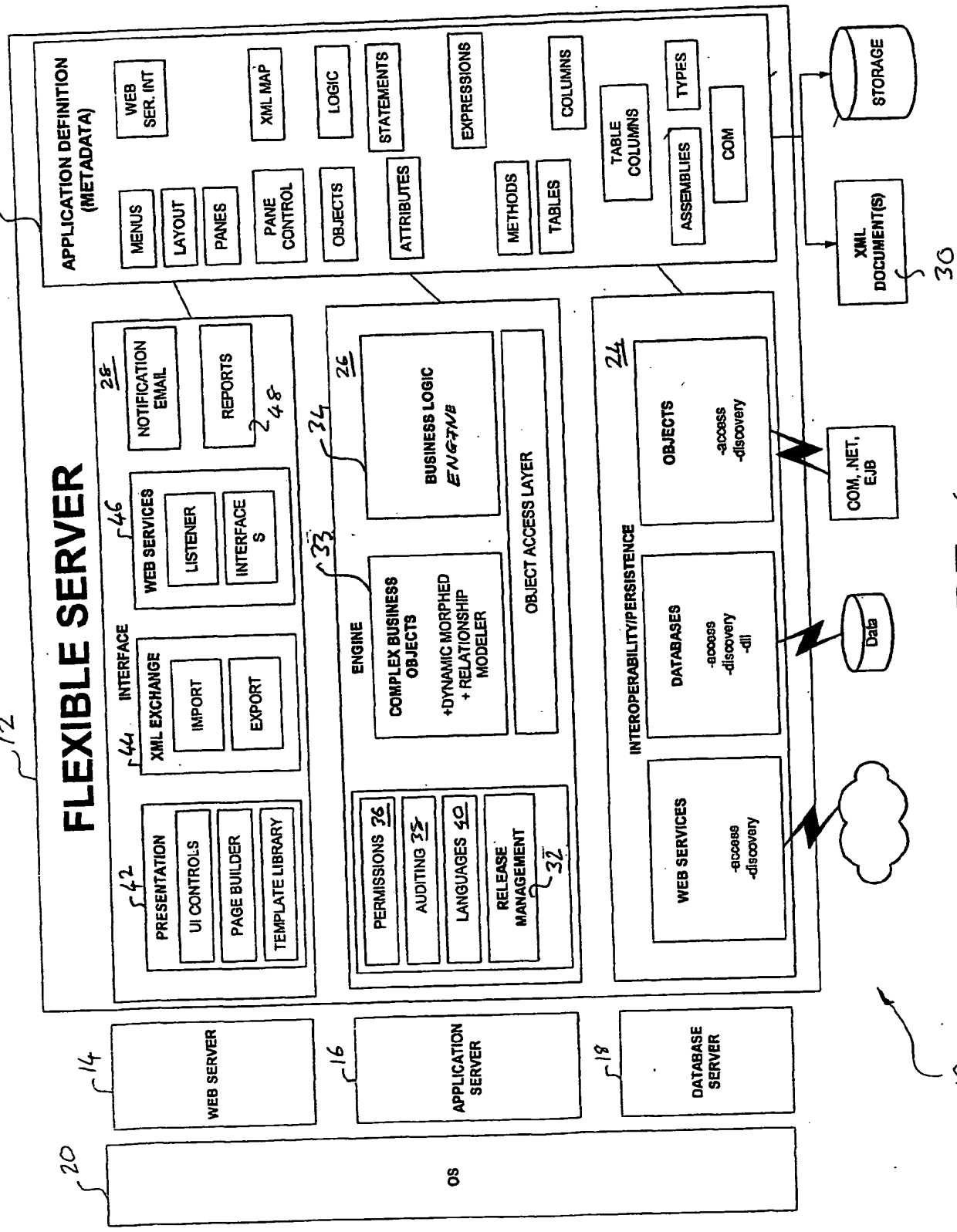

FLEXIBLE SERVER
12
APPLICATION DEFINITION (METADATA)
22
MENUS
LAYOUT
PANES
PANE CONTROL
OBJECTS
ATTRIBUTES
WEB SER. INT
XML MAP
LOGIC
STATEMENTS
EXPRESSIONS
METHODS
TABLES
COLUMNS
TABLE COLUMNS
ASSEMBLIES
TYPES
COM
STORAGE
XML DOCUMENT(S)
30
INTERFACE
XML EXCHANGE
IMPORT
EXPORT
44
PRESENTATION
UI CONTROLS
PAGE BUILDER
TEMPLATE LIBRARY
42
WEB SERVICES
LISTENER
INTERFACES
46
NOTIFICATION EMAIL
28
REPORTS
48
ENGINE
33
COMPLEX BUSINESS OBJECTS
+DYNAMIC MORPHED + RELATIONSHIP MODELER
BUSINESS LOGIC ENGINE
34
26
OBJECT ACCESS LAYER
PERMISSIONS 36
AUDITING 38
LANGUAGES 40
RELEASE MANAGEMENT 32
INTEROPERABILITY/PERSISTENCE
24
OBJECTS
-access
-discovery
DATABASES
-access
-discovery
-dll
WEB SERVICES
-access
-discovery
COM, .NET, EJB
Data
WEB SERVER
14
APPLICATION SERVER
16
DATABASE SERVER
18
OS
20
10

FIGURE 1

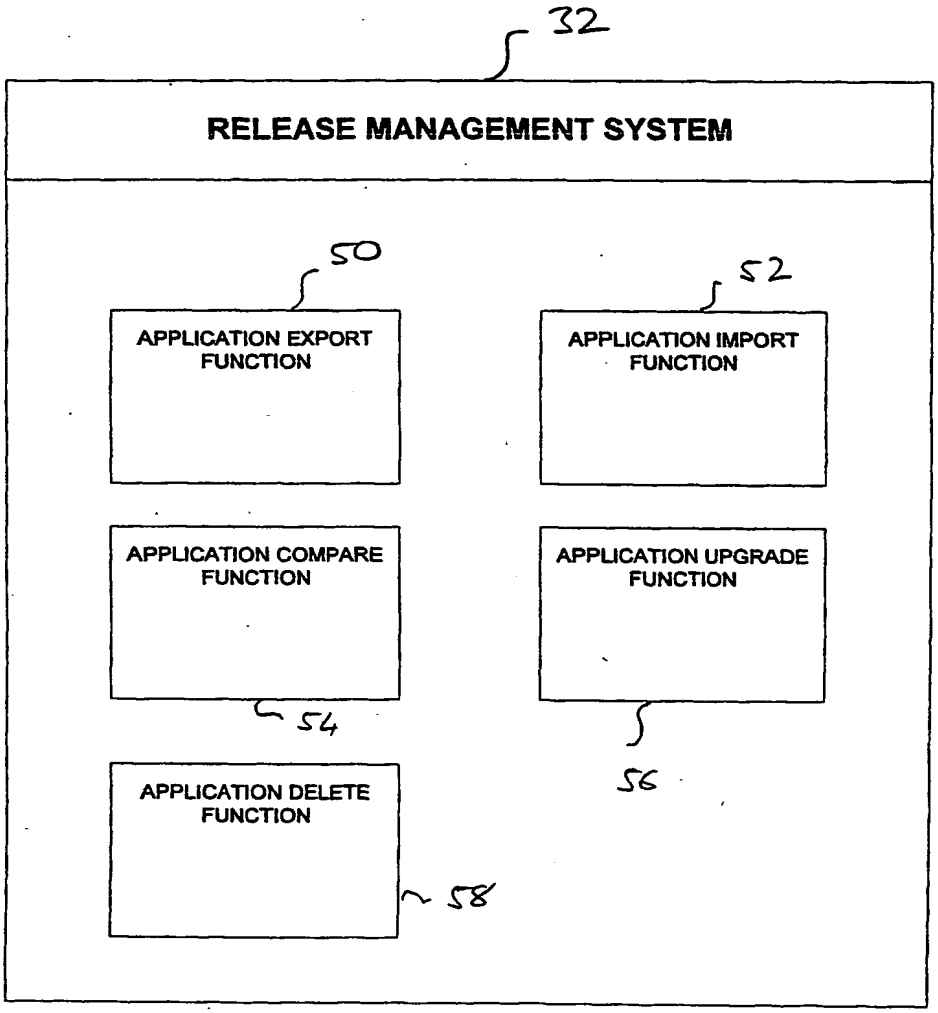

32
RELEASE MANAGEMENT SYSTEM
50
APPLICATION EXPORT FUNCTION
52
APPLICATION IMPORT FUNCTION
APPLICATION COMPARE FUNCTION
54
APPLICATION UPGRADE FUNCTION
56
APPLICATION DELETE FUNCTION
58

FIGURE 2

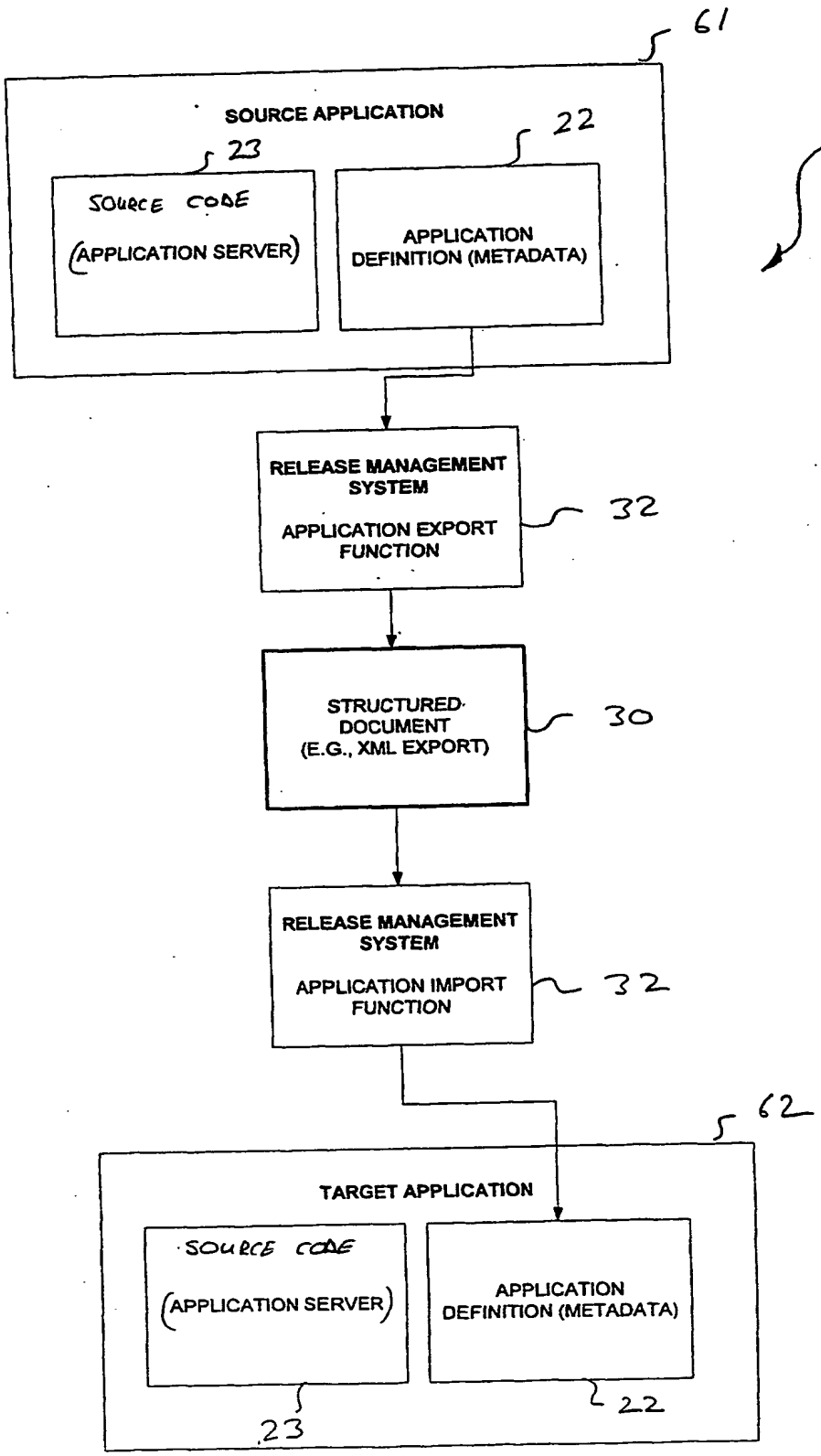

61
SOURCE APPLICATION
23
22
60
SOURCE CODE
(APPLICATION SERVER)
APPLICATION DEFINITION (METADATA)
RELEASE MANAGEMENT SYSTEM
APPLICATION EXPORT FUNCTION
32
STRUCTURED DOCUMENT (E.G., XML EXPORT)
30
RELEASE MANAGEMENT SYSTEM
APPLICATION IMPORT FUNCTION
32
62
TARGET APPLICATION
SOURCE CODE
(APPLICATION SERVER)
APPLICATION DEFINITION (METADATA)
23
22

FIGURE 3

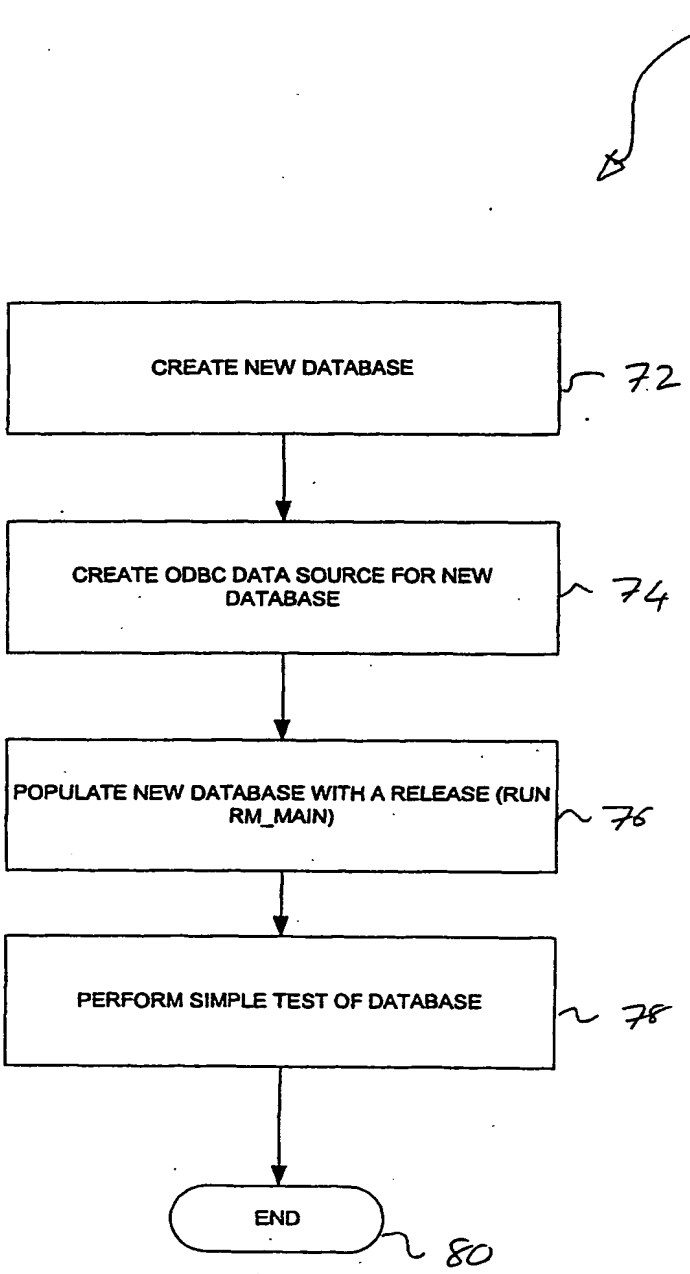


FIGURE 4

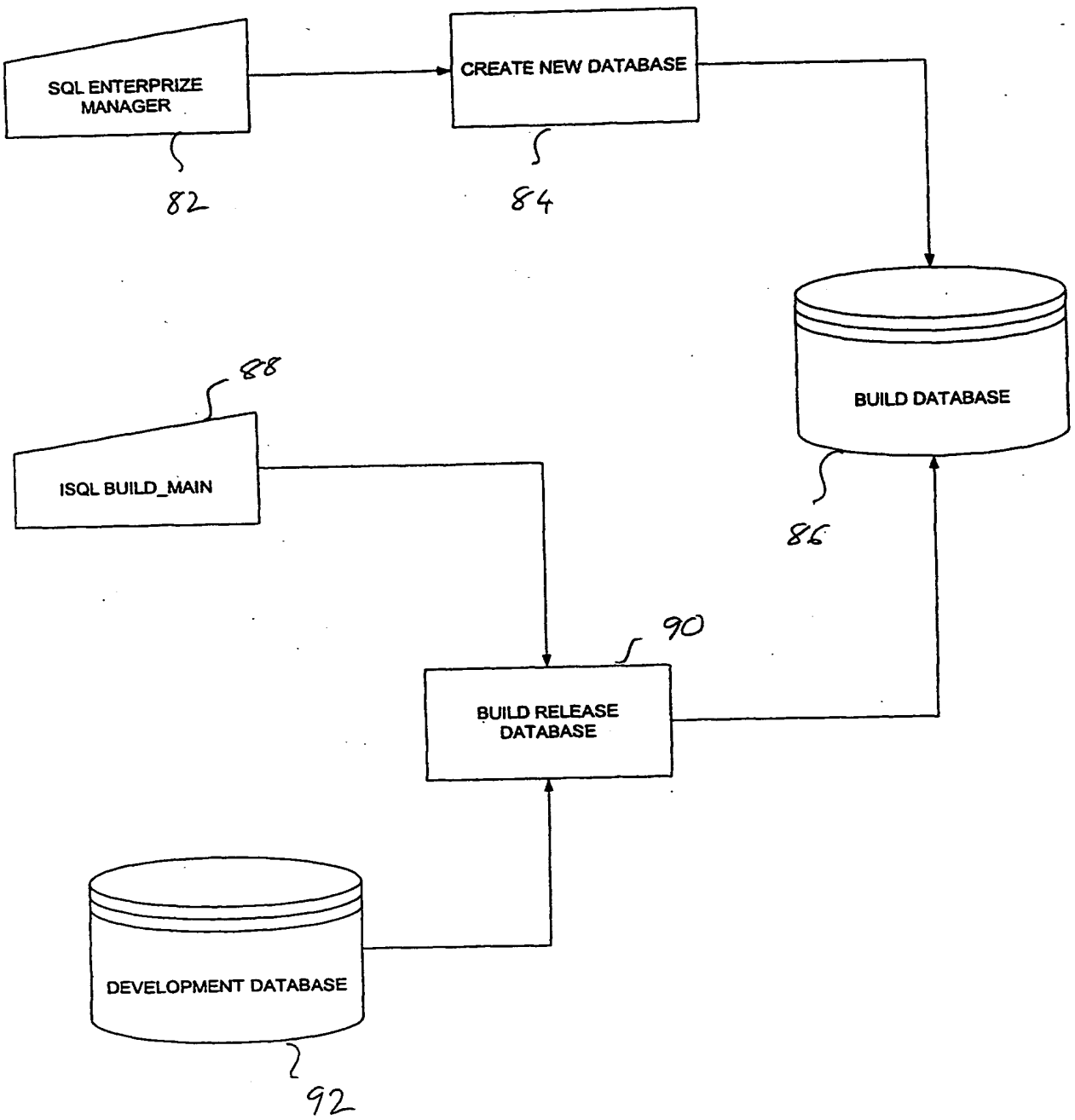

SQL ENTERPRIZE MANAGER
82
CREATE NEW DATABASE
84
BUILD DATABASE
86
88
ISQL BUILD_MAIN
90
BUILD RELEASE DATABASE
DEVELOPMENT DATABASE
92

FIGURE 5

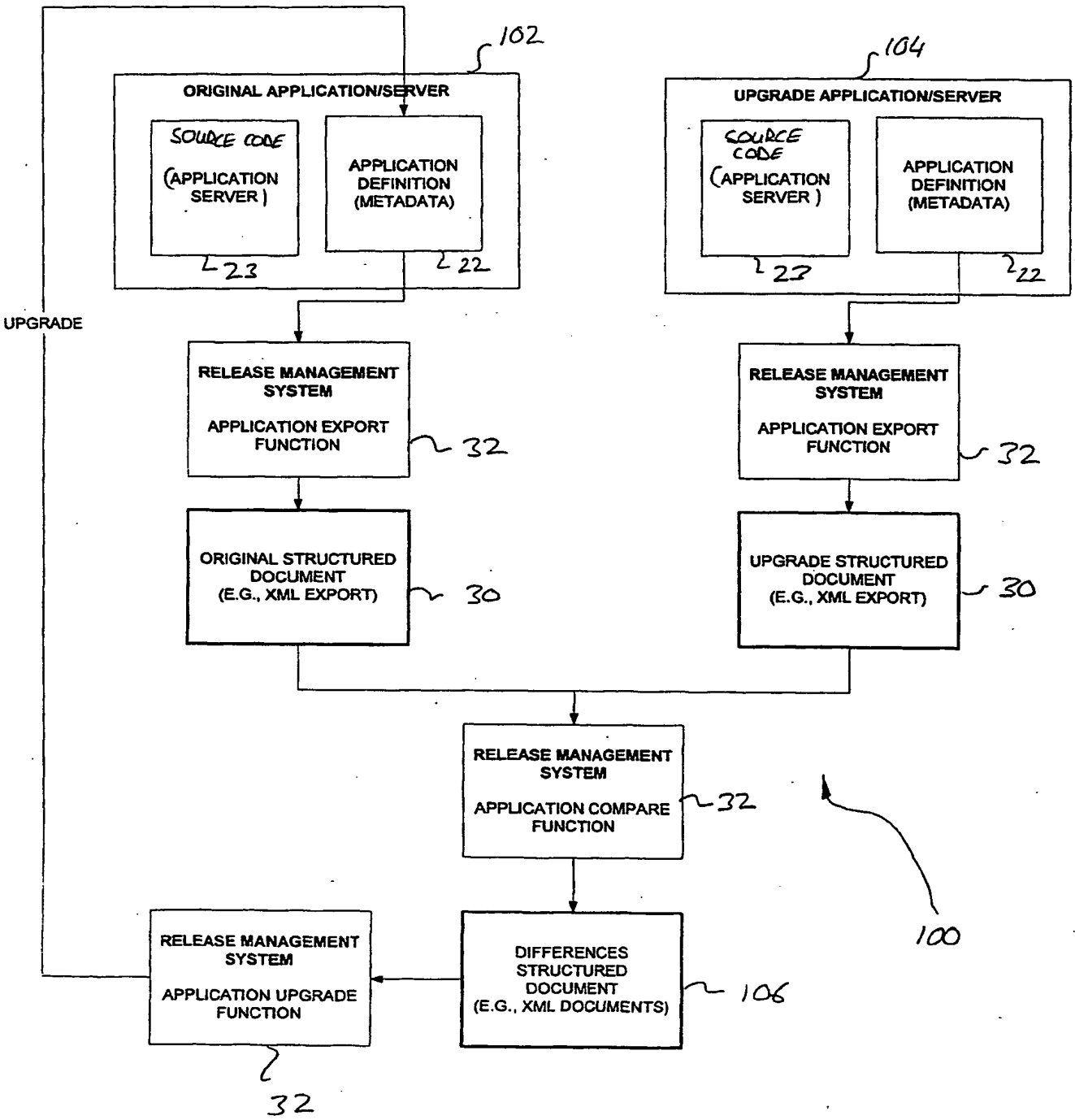

102
ORIGINAL APPLICATION/SERVER
SOURCE CODE (APPLICATION SERVER)
APPLICATION DEFINITION (METADATA)
23
22
104
UPGRADE APPLICATION/SERVER
SOURCE CODE (APPLICATION SERVER)
APPLICATION DEFINITION (METADATA)
23
22
UPGRADE
RELEASE MANAGEMENT SYSTEM
APPLICATION EXPORT FUNCTION
32
RELEASE MANAGEMENT SYSTEM
APPLICATION EXPORT FUNCTION
32
ORIGINAL STRUCTURED DOCUMENT (E.G., XML EXPORT)
30
UPGRADE STRUCTURED DOCUMENT (E.G., XML EXPORT)
30
RELEASE MANAGEMENT SYSTEM
APPLICATION COMPARE FUNCTION
32
100
RELEASE MANAGEMENT SYSTEM
APPLICATION UPGRADE FUNCTION
32
DIFFERENCES STRUCTURED DOCUMENT (E.G., XML DOCUMENTS)
106

FIGURE 6

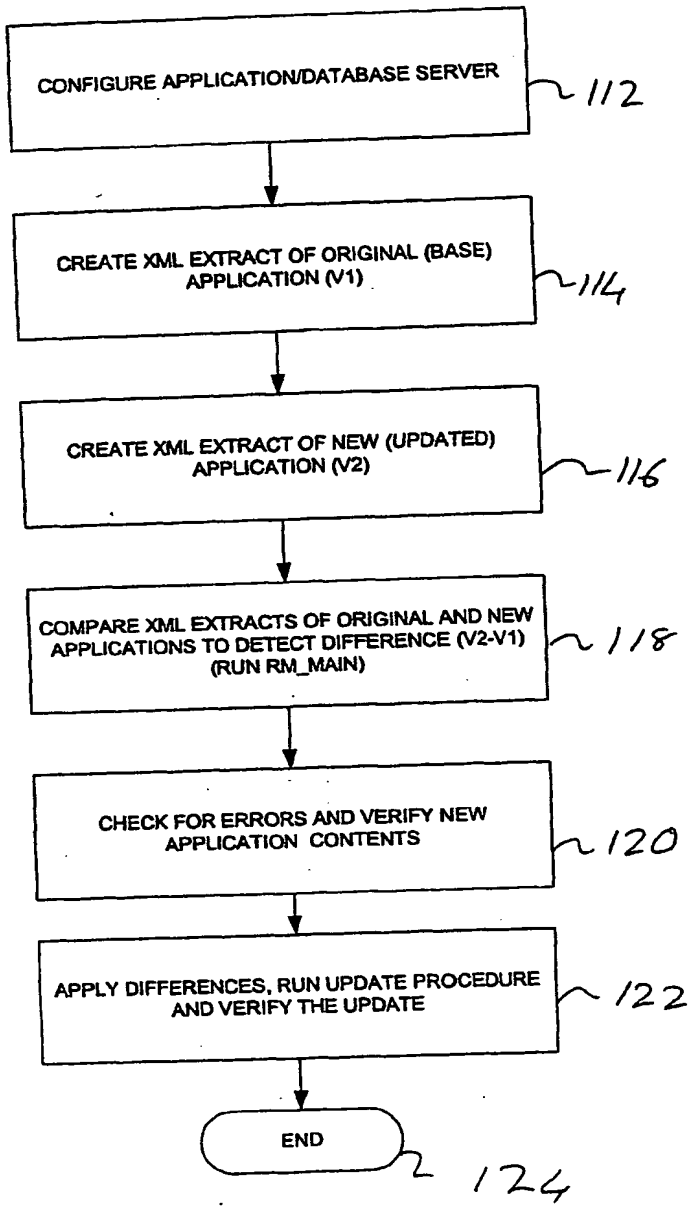

CONFIGURE APPLICATION/DATABASE SERVER
112
110'
CREATE XML EXTRACT OF ORIGINAL (BASE) APPLICATION (V1)
114
CREATE XML EXTRACT OF NEW (UPDATED) APPLICATION (V2)
116
COMPARE XML EXTRACTS OF ORIGINAL AND NEW APPLICATIONS TO DETECT DIFFERENCE (V2-V1) (RUN RM_MAIN)
118
CHECK FOR ERRORS AND VERIFY NEW APPLICATION CONTENTS
120
APPLY DIFFERENCES, RUN UPDATE PROCEDURE AND VERIFY THE UPDATE
122
END
124

FIGURE 7

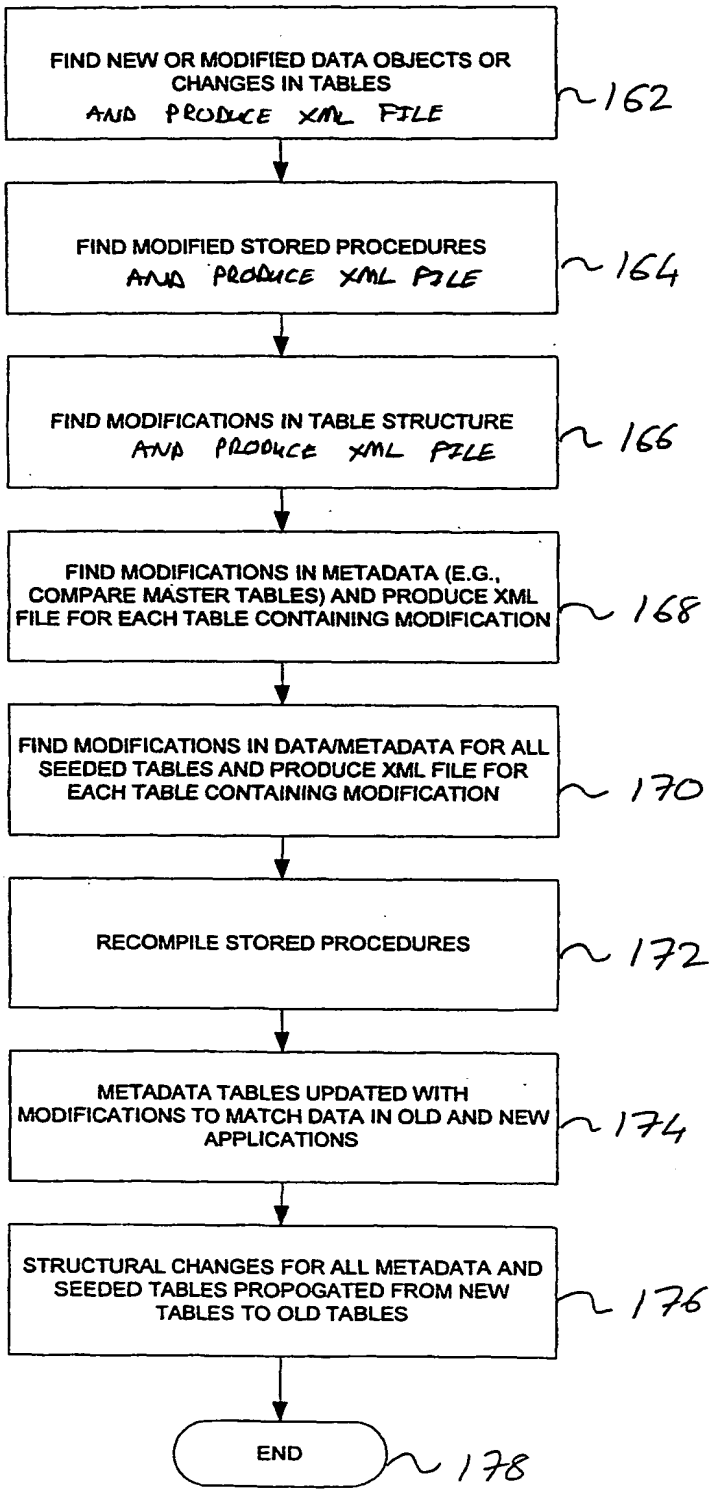

FIND NEW OR MODIFIED DATA OBJECTS OR CHANGES IN TABLES AND PRODUCE XML FILE
162
160
FIND MODIFIED STORED PROCEDURES AND PRODUCE XML FILE
164
FIND MODIFICATIONS IN TABLE STRUCTURE AND PRODUCE XML FILE
166
FIND MODIFICATIONS IN METADATA (E.G., COMPARE MASTER TABLES) AND PRODUCE XML FILE FOR EACH TABLE CONTAINING MODIFICATION
168
FIND MODIFICATIONS IN DATA/METADATA FOR ALL SEEDED TABLES AND PRODUCE XML FILE FOR EACH TABLE CONTAINING MODIFICATION
170
RECOMPILE STORED PROCEDURES
172
METADATA TABLES UPDATED WITH MODIFICATIONS TO MATCH DATA IN OLD AND NEW APPLICATIONS
174
STRUCTURAL CHANGES FOR ALL METADATA AND SEEDED TABLES PROPOGATED FROM NEW TABLES TO OLD TABLES
176
END
178

FIGURE 8

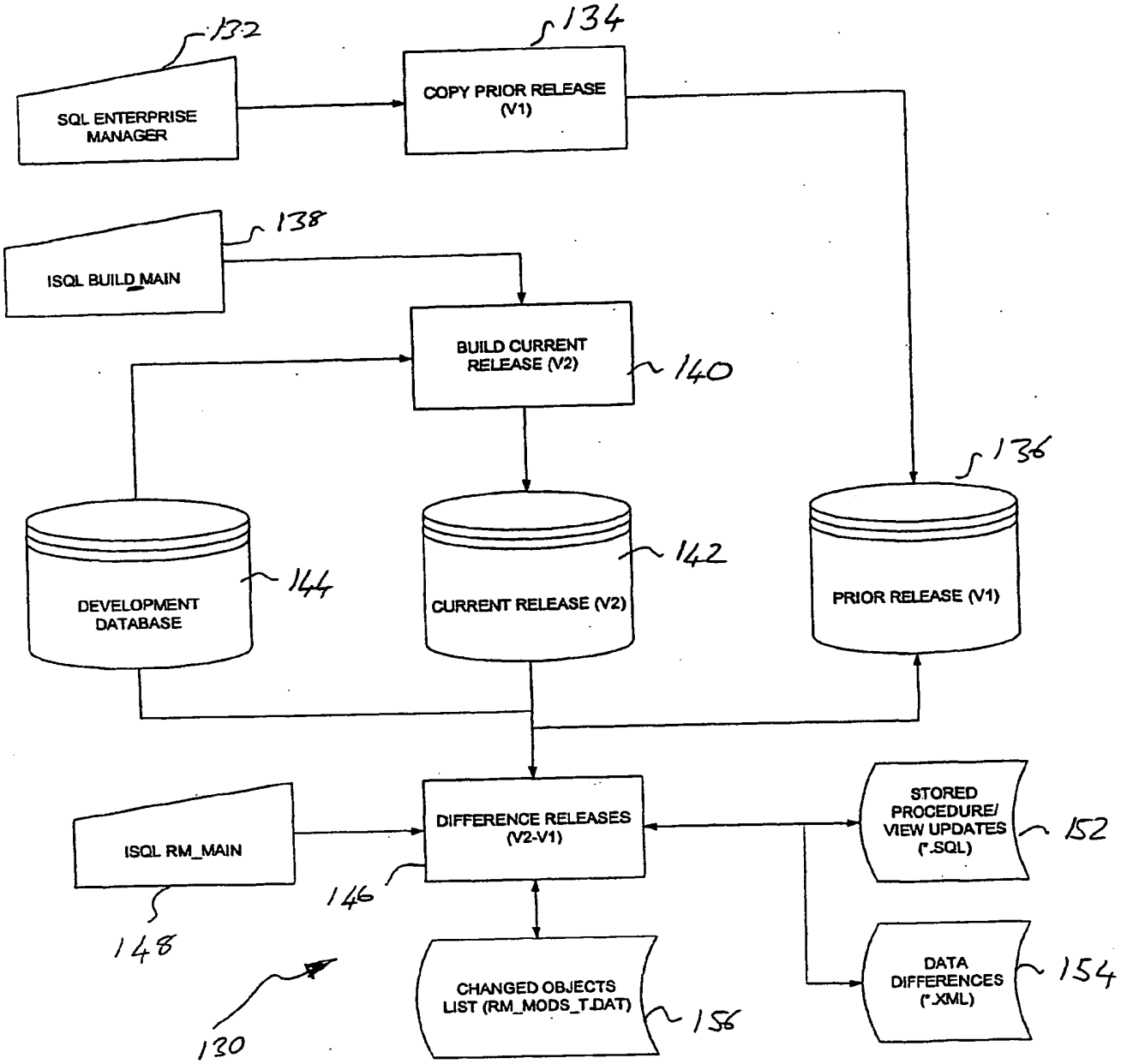

132
SQL ENTERPRISE MANAGER
134
COPY PRIOR RELEASE (V1)
138
ISQL BUILD_MAIN
BUILD CURRENT RELEASE (V2)
140
136
DEVELOPMENT DATABASE
144
CURRENT RELEASE (V2)
142
PRIOR RELEASE (V1)
ISQL RM_MAIN
DIFFERENCE RELEASES (V2-V1)
STORED PROCEDURE/ VIEW UPDATES (*.SQL)
152
146
148
DATA DIFFERENCES (*.XML)
154
CHANGED OBJECTS LIST (RM_MODS_T.DAT)
156
130

FIGURE 9

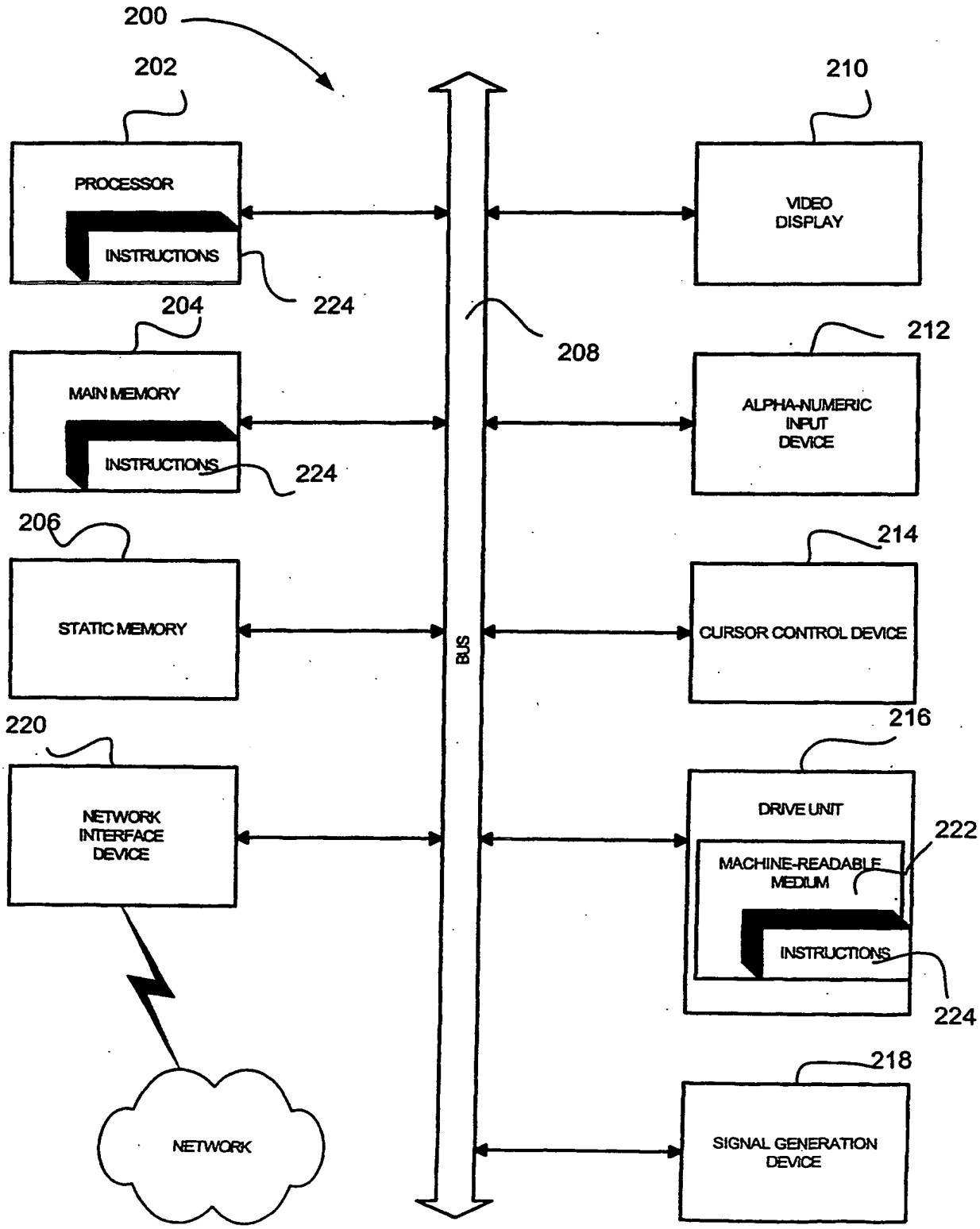


FIGURE 10

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US01/47932

A. CLASSIFICATION OF SUBJECT MATTER
IPC(7) : G06F 7/02
US CL : 707/501.1, 513; 717/118, 168
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
U.S. : 707/501.1, 513; 717/118, 168

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
USPAT, USPG-PUB, EPO, JPO, DERWENT, IBM TDB

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 6,052,711 A (GISH) 18 April 2000, column 15, line 1-column 17, line 67.	1-30
A	US 6,038,590 A (GISH) 14 March 2000, column 14, line 35-column 18, line 67.	1-30
A	US 5,999,972 A (GISH) 07 December 1999, column 14, line 35-column 18, line 67.	1-30
A	US 5,835,914 A (BRIM) 10 November 1998, column 2, line 11-67.	1-30
A	US 5,995,756 A (HERRMANN) 30 November 1999, column 3, line 1-column 4, line 16.	1-30
A	US 6,003,065 A (YAN et al) 14 December 1999, column 23, line 1-column 24, line 67.	1-30
A	US 6,125,363 A (BUZZEO et al) 26 September 2000, column 2, line 1-line 67.	1-30

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search
26 March 2002 (26.03.2002)

Date of mailing of the international search report
22 APR 2002

Name and mailing address of the ISA/US
Commissioner of Patents and Trademarks
Box PCT
Washington, D.C. 20231
Facsimile No. (703)305-3230

Authorized officer
STEPHEN HONG

Telephone No. (703) 305-3900

Form PCT/ISA/210 (second sheet) (July 1998)

INTERNATIONAL SEARCH REPORT

International application No.

PCT/US01/47932

Continuation of Item 4 of the first sheet:
AUTOMATICALLY DEPLOY AND UPGRADE AN APPLICATION BASED ON AN MARKUP LANGUAGE APPLICATION DEFINITION